**T1**

Statement of Intent

Overview

This document aims to provide a record of static code analysis performed on a specific issue from the Coverity SAST scan for the NASA ION Open-Source code 4.1.1 project.

The primary purpose of this document is to validate the issue identified via the automated detection process to eliminate false positives.

Depending on findings, secondary purposes can include but are not limited to listing/providing recommended fixes alongside a list of attack vectors and potential exploits for consideration.

Reporting Best Practices

Please ensure best practices are kept when completing the document via regularly updating the Acronyms and Abbreviations table alongside any iterations made to the Document History table. This will allow other members to identify any updates and progress made across trimesters easily.

When using code snippets, please use screenshots that are clear and easy to read, alternatively, use words built-in code formatter found [here](https://appsource.microsoft.com/en-us/product/office/WA104382008?tab=Overview).

Document Naming Conventions

Naming conventions for this file are as follow; SAR\_{CID}. For example, when investigating issue 123456 the file name would be SAR\_123456.docx

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Dates** | **Version** | **Author** | **Comments** |
| 24/03/2023 | V0.1 | Amber Smith |  |
|  |  |  |  |
|  |  |  |  |

Table of Content

Contents

[Introduction 3](#_Toc119848724)

[Objective 3](#_Toc119848725)

[Scope 3](#_Toc119848726)

[Acronyms and Abbreviations 3](#_Toc119848727)

[Code Review and Analysis 4](#_Toc119848728)

[Outcomes 4](#_Toc119848729)

[Observations 4](#_Toc119848730)

[Supporting Evidence 4](#_Toc119848731)

[Conclusions and Recommendations 4](#_Toc119848732)

[References 5](#_Toc119848733)

[Appendix 6](#_Toc119848734)

# Introduction

## Objective

The primary objective of this analysis is to determine whether the defects identified in the Coverity Report for the ION Open Source 4.1.1 project are:

* Indeed, defects.
* Potentially exploitable.

The secondary objective of this analysis, where applicable, is to provide the following:

* Recommendation(s) to fix.
* Any exploit for consideration.

## Scope

This static code analysis is limited to the out-of-bounds access type defect identified in the following CIDs:  
CID1520837

# Acronyms and Abbreviations

Please keep an updated list of acronyms and abbreviations used throughout the report.

|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| DTN | Delay/Disruption Tolerant Network |
| ION | Interplanetary Overlay Network |
|  |  |

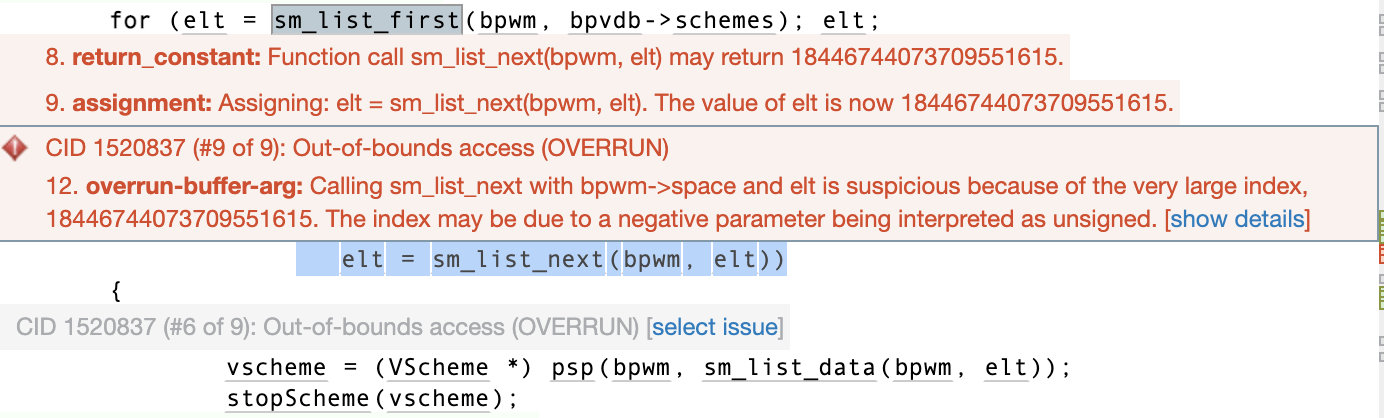
# Code Review and Analysis

## 

## Outcomes

Coverity identified this CID as having an out-of-bounds access error. This is also known as a buffer overflow. Buffer overflows occur when data stored inside the buffer is outside the range of allocated memory. They have the ability to read and write outside of the boundaries intended.

## Observations

**

This line of code is using a loop to go through a linked list (bpvdb->schemes) of elements. It is using the sm\_list\_first and sm\_list\_next functions to go through. The loop is using the variable elt and executes the variable on the first element of the list and the continues throught the list doing the same and updating the elt variable prior to moving onto the next element. When the end of the loop is reached it returns NULL, making the loop terminate.

The error is telling that there is a suspicious call to the sm\_list\_data function and that an elt index is very large (18446744073709551615 is the maximum value of a 64-bit unsigned integer). This message indicates that there might be an issue regarding the interpretation of a negative parameter as an unsigned value. This can result in the index being set to a very large value. If the signedness of a parameter is not specified, there can be incorrect behaviour exhibited when the parameter is used in a calculation.

This means that there is a risk of accessing memory that exists beyond an allocated buffer due to the overflow issue. Negative interger values can be interpreted as unsigned, meaning the maximum value possible (ie. 18446744073709551615) will be used as the index.

# Conclusions and Recommendations

Please provide any conclusions and recommendations here. Include references to supporting evidence and any other required information in the appendix.

To fix this issue, the code needs to be reviewed so that when it calculates the index value it needs to be correctly interpreted as either signed or unsigned as an integer type. To do this, we can use the line:

uint64\_t index = (uint64\_t)(intptr\_t)elt; //for when elt is expected to be a signed integer.

uint64\_t index = (uint64\_t)elt; //for when elt is expected to be an unsigned integer.

Additionally, implementing bounds checking would assist in the prevention of buffer overruns.